

# A General Meta-Heuristic Based Solver for Combinatorial Optimisation Problems

Marcus Randall \*

School of Information Technology  
Bond University, QLD 4299, Australia

David Abramson

School of Computer Science and Software Engineering  
Monash University, VIC 3168, Australia

March 23, 2000

## Abstract

In recent years, there have been many studies in which tailored heuristics and meta-heuristics have been applied to specific optimisation problems. These codes can be extremely efficient, but may also lack generality. In contrast, this research focuses on building a *general-purpose* combinatorial optimisation problem solver using a variety of meta-heuristic algorithms including Simulated Annealing and Tabu Search. The system is novel because it uses a modelling environment in which the solution is stored in dense dynamic list structures, unlike a more conventional sparse vector notation. Because of this, it incorporates a number of neighbourhood search operators that are normally only found in tailored codes and it performs well on a range of problems. The general nature of the system allows a model developer to rapidly prototype different problems. The

---

\*We would like to thank the Queensland Parallel Supercomputing Facility for the use of the IBM SP2 computer. This work was supported by an Australian Research Council Large Grant. Thanks go to Dr Clyde Wild from Griffith University School of Environmental and Applied Science for his assistance with the statistical evaluation of the solver.

new solver is applied across a range of traditional combinatorial optimisation problems. The results indicate that the system achieves good performance in terms of solution quality and runtime.

**Keywords:** Combinatorial Optimisation, Meta-heuristic search algorithms, Linked lists

## 1 Introduction

There have been many studies in which heuristic and meta-heuristic search algorithms have been applied to Combinatorial Optimisation Problems (COPs). In many cases, these have been remarkably successful in finding optimal and near-optimal solutions efficiently. A large part of this success is due to careful tailoring and tuning of the algorithm and data structures for particular problems and problem instances which can take substantial development time. Relatively little research has been performed to produce general heuristic based problem solvers, apart from [6] [20] [29] [30] [32] [33] [37], as it is a difficult task and it is thought that such an approach would be too inefficient. In this work, we examine the issue of modelling representations for COPs and develop a system based on densely packed dynamic data structures (in particular linked lists). The general-purpose solver based on this modelling representation is shown to have good performance on a wide range of COPs.

The traditional way of expressing COPs mathematically has been to use a sparse 0-1 Integer Linear Programme (ILP) vector formulation. In this model, variables take on either a value of 0 or 1 to denote assignment of items to groups. This is a very general and flexible approach that has been quite successful for a range of ILPs, particularly for finding lower and upper bounds on the optimal solution. However, in some cases it is difficult to map the structure of a COP onto a 0-1 space, resulting in a sparse problem model that contains a large number of variables and constraints. This is especially evident for problems with a complex underlying structure such as the Travelling Salesman Problem (TSP) in which a set of cities is visited but only with a Hamiltonian tour.

Search algorithms for these problems may be broadly classified into *exact* and *non-exact* approaches. The first of which consists

of techniques from Operations Research (OR) and to a lesser extent Artificial Intelligence (AI). These techniques (such as branch and bound and branch and cut [53] for OR and Constraint Logic Programming [31] [40] [57] and A\* [27] for AI) aim to produce solutions that have proven optimality. While COPs can be solved in this way, many are NP hard and as such can require exponential computational time to solve to proven optimality. An alternative group of techniques, known as the *heuristics*, take a different approach to solving COPs. Instead of seeking proven optimality, they endeavour to find good (near-optimal) solutions in reasonable amounts of computational time. A heuristic is applied to a specific problem and exploits that problem's structural properties in order to find solutions. Whilst these are often very efficient (see for example the TSP heuristic in [39]), they cannot be readily adapted to other problem types. However, there is a class of heuristic techniques that are applicable across problem type and these are known as the *meta-heuristics*. Well-known meta-heuristic techniques are Simulated Annealing (SA) [35] [58], Tabu Search (TS) [24], Greedy Search, Greedy Randomised Adaptive Search (GRASP) [22] [24], Genetic Algorithms (GAs) [25] and Artificial Neural Networks (ANNs) [28] [56].

There have been few attempts made in the past to produce a general meta-heuristic system for COPs. This is due mainly to two reasons. First, it is difficult to build a general-purpose meta-heuristic, as this requires data structures that can store information about a wide range of problems as well as the application of appropriate transition operators for particular problems. Therefore, the common practice in the past has been to incorporate structural information about the problem within the meta-heuristic search algorithm. Some examples from the literature include [11] [12] [18] [19] [34] [42] [43] [50] [54] [55]. Second, the most general modelling representation for integer optimisation problems is based on 0-1 ILP notation [53], however, meta-heuristics find it difficult to navigate the sparse 0-1 space of a COP. This is because changing a value of a variable can break many of the problem's constraints, and restoring these to a feasible state is a costly exercise (especially for practical size problems) [4] [6].

We have found that there are two basic approaches to producing general-purpose meta-heuristic codes. The first of which is characterised by GPSIMAN (General Purpose SIMulated ANnealing) [20], that solves 0-1 ILPs with SA. This approach is adequate for very small

problems only, because of the 0-1 encoding. The feasibility restoration process that it employs is based on a degenerative tree search in which certain variables are flipped in order to return to a feasible state. However, this process is the most computationally intense part of the entire code and as such GPSIMAN was shown to be impractical for problems larger than the original test set [6].

Johnson et al.'s [32] [33] and Ingber's [29] [30] approach differs from GPSIMAN as these systems can be tailor-made to solve a particular problem from component libraries. Their systems consist of a core SA engine for which the developer is required to provide an interface (via code modules) for a particular problem. This specifies the cost, constraint and transition functions. These systems are not based on an algebraic modelling language and while code development is a flexible approach to modelling a particular problem, it can require the developer to spend a substantial amount of time defining the problem to be solved. However, the tailor-made code is more efficient than algebraic based solvers such as GPSIMAN. Skeleton GA codes are also available from Bäck [7] and Grefenstette [26]. Because GAs have a standard set of genetic transition operators, only a single cost function and problem data needs to be specified for each problem.

The aim of this research is to develop a general-purpose problem solver that is based on meta-heuristic search techniques. This is achieved in two stages. First, a new algebraic modelling representation is proposed that is exclusively designed for COPs. We also show that the modelling representation allows the direct application of local search transition operators, hence making it suitable for meta-heuristic algorithms. Second, a solver based on the list modelling representation is built and is shown to have good performance on a range of problems.

The remainder of this paper is organised as follows. Section 2 describes the new linked list modelling representation for COPs. Section 3 describes the application of local search transition operators given this new representation. From this, the implementation of a general COP solver is shown in Section 4. The testing of the solver is described in Section 5. Finally, conclusions are drawn in Section 6.

## 2 Modelling COPs Using Linked Lists

The dynamic data structure, the list, has been shown to be useful for performing general computation as illustrated by the computer programming language LISP (LISt Processing). Lists are dense structures in which elements are linked together in a linear fashion. Lists can shrink and grow by adding and subtracting elements respectively. An element in a list can itself be a list (i.e. a sub-list). We have found that this structure is useful for expressing mappings or grouping of items as the primary purpose of combinatorial optimisation is “[to] find an optimal arrangement, grouping, ordering or selection of discrete objects usually finite in number” [44], p. 2. As a result of the mappings being expressed in terms of lists, the elements in the list can take on a range of integer values. Subsequently, many of the encoding constraints associated with other modelling representations (particularly 0-1 ILPs) are eliminated. Another advantage is that a solution expressed as a list can be altered by using common local search operators (discussed in Section 3).

### 2.1 Properties of a List Modelling Representation

In this section, a general model is developed that specifies COPs as dense linked list structures. In general, a list can be nested recursively to an arbitrary depth, and thus a list can contain either data values or pointers to other lists. However, in all of the problems we have addressed, the recursion required is only one level. Consequently, in this document, the main list consists only of sub-lists or data elements. The placement of simple elements on particular sub-lists or at certain positions on the sub-lists defines the actual solution to the problem. To denote that element  $e$  is assigned to the  $i^{th}$  sub-list at the  $j^{th}$  position, the notation  $x(i, j) = e$  is used. In this work, we concentrate on problems that can be expressed with one decision list (denoted by  $x$ ). For instance, consider a TSP in which city 5 is assigned to position 2 therefore  $x(2) = 5$ . This means that city 5 is the second stop on the salesman’s tour. All other literals in the problem models are considered as constants, vectors or matrices as appropriate.

The structure of the list is defined for each particular problem and is characterised by:

- The number and location of sub-lists within the overall list struc-

ture.

- The size of the lists (specified in terms of a lower and upper bound).
- The range of values that may appear on a list.
- The number of times that a particular value can appear on a list.

The general list modelling representation is stated mathematically according to Equations 1 - 7.

$$\text{Optimise } f(x) \quad (1)$$

*s.t.*

$$lhs_i(x) \begin{pmatrix} = \\ < \\ \leq \\ > \\ \geq \\ \neq \end{pmatrix} rhs_i \quad 1 \leq i \leq C \quad (2)$$

$$l_y \begin{pmatrix} = \\ < \\ \leq \end{pmatrix} |y| \begin{pmatrix} = \\ < \\ \leq \end{pmatrix} u_y \quad (3)$$

$$min\_count(y) \begin{pmatrix} = \\ > \\ \geq \end{pmatrix} m_y \quad (4)$$

$$max\_count(y) \begin{pmatrix} = \\ < \\ \leq \end{pmatrix} n_y \quad (5)$$

$$p_y \leq E_y \leq q_y \quad (6)$$

$$E_x \subset \mathbb{N} \quad (7)$$

Where:

$x$  is the decision list. For instance:  $x(i, j)$  is the  $j^{th}$  element value of the  $i^{th}$  sub-list.

$f(x)$  is the objective function applied to decision list  $x$ .

$lhs_i(x)$  is the left-hand side of problem constraint  $i$  applied to decision list  $x$ .

$rhs_i$  is the right-hand side of problem constraint  $i$ .

$C$  is the number of problem constraints.

$y$  represents any list structure within list  $x$ .

$|y|$  is the length of the list defined by  $y$ .

$l_y$  is the minimum length of the list  $y$ .

$u_y$  is the maximum length of list  $y$ .

$min\_count(y)$  and  $max\_count(y)$  return the number of occurrences of the least frequent and most frequent element value on list  $y$  respectively. These functions are discussed in detail below.

$m_y$  defines the minimum number of occurrences of each value on list  $y$ .

$n_y$  defines the maximum number of occurrences of each value on list  $y$ .

$E_y$  represents the set of element values contained on list  $y$ .

$p_y$  is the smallest value that can be contained in the list  $y$ .

$q_y$  is the largest value that can be contained in the list  $y$ .

$\mathbb{N}$  is the set of Natural Numbers.

Using this modelling representation, constraints are conceptually grouped into two distinct classes, namely the list constraints (constraints that govern the list structure) and the *problem constraints* (those that define the feasible space of the problem). Equations 3 - 7 in the preceding model refer to the list constraints, while 2 represents the problem constraints. In general, list constraints can be divided into three categories; *Value Range*, *Count* and *List Size*.

The legal values of elements in a particular list are integral and specified between a lower and upper bound. This category is referred to as the *Value Range* and is in the form of Equations 6 and 7. For instance, if a list stores the order of the cities for an  $N$  city TSP, the value range would be defined between 1 and  $N$ .

The *Count* constraints regulate the number of occurrences of each value on a list. They are specified using the functions  $min\_count$  and  $max\_count$  (Equations 4 and 5).  $min\_count(y)$  returns the cardinality of the element that appears least on list  $y$ . For instance, given that

list  $y$  consists of the elements  $\{5, 5, 6, 3, 3\}$ ,  $\text{min\_count}(y)$  returns 1 as the element value 6 appears once in the list.  $\text{max\_count}$  is the opposite is this as it returns the cardinality of the element that appears most on list  $y$ . In the previous example,  $\text{max\_count}(y)$  returns 2. Permutations of values are widely used in COPs and can be specified using the constraints  $\text{min\_count}(y) = 1$  and  $\text{max\_count}(y) = 1$ . This combination limits each value to one appearance on the list. In the case of the TSP, these constraints allow only tours that do not violate the condition that each city is only visited once. The *Count* constraint type can eliminate many of the complex encoding constraints required for other representations (especially 0-1 ILPs).

The size and shape of a list structure is specified using equations of the form 3. This category is referred to as the *List Size*. Using this notation, an arbitrary list structure can be built as well as allowing the size of a list to be either static or dynamic. If the latter is the case, the size of the list may vary between preset lower and upper limits or without limit on the size.

The list modelling representation is very general and is capable of representing a remarkable variety of COPs including the travelling salesman problem, quadratic assignment problem, graph problems, timetabling problems and scheduling problems. For a thorough list of problems and their list-based models, see [46].

## 2.2 A Practical Example of a List Model

Using the general model, consider the Generalised Assignment Problem (GAP) [18] [43]. In this problem, jobs are assigned to agents for these agents to perform subject to capacity constraints. The linked list representation associates agents with sub-lists and these sub-lists contain a number of jobs. Figure 1 is an example of a 5 agent, 15 job problem. In this example, job 4 is assigned to agent 3 and job 13 is assigned to agent 5 etc.

*Desired location of Figure 1*

The objective function is to minimise the total cost of assigning the jobs to the agents and can be expressed by 8.

$$\sum_{i=1}^M \sum_{j=1}^{|x(i)|} C(i, x(i, j)) \quad (8)$$

Where:

$x$  is the solution list.  $x(i, j)$  is the  $j^{th}$  job performed by agent  $i$ .

$C$  is the cost matrix.  $C(i, j)$  is the cost of assigning job  $j$  to agent  $i$ .

$M$  is the number of agents.

Whilst this list notation appears similar to conventional vector form used in standard LPs, each sub-list contains a variable number of elements. Thus, the second summation sign in 8 requires a bound which varies depending on the length of each sub-list (i.e.  $|x(i)|$ ) and changes according to the current solution state. Similarly, the constraints concerning the capacity of each agent are formed across list space. In this example, Equations 10 - 13 are the list constraints while 9 represents the problem constraints.

$$\sum_{j=1}^{|x(i)|} a(i, x(i, j)) \leq b(i) \quad \forall i \quad 1 \leq i \leq M \quad (9)$$

$$|x| = M \quad (10)$$

$$1 \leq x(i, j) \leq N \quad \forall i, j \quad 1 \leq i \leq M \quad 1 \leq j \leq |x(i)| \quad (11)$$

$$\min\_count(x) = 1 \quad (12)$$

$$\max\_count(x) = 1 \quad (13)$$

Where:

$a$  is the resource matrix.  $a(i, j)$  is the resource required by agent  $i$  to perform job  $j$ .

$b$  is the capacity vector.  $b(i)$  is the capacity of agent  $i$ .

$N$  is the number of jobs.

### 3 Search Over Lists

The advantage of using the list representation scheme is that well-known local search operators (that have successfully been used in other heuristic codes, for example [1] [15] [20] [32] [33] [34] [42] [43] [54]) can be applied directly to the list structure. There are a variety of ways that a list can be altered in order to form a new solution. However, seven different transition operators have been identified as sufficient to navigate the search space of the list-based formulations. These operators each only make a small change to the solution. In this study, the emphasis is placed on simple yet functional operators that can either be used separately or in combination with one another to achieve good quality solutions. Initial empirical investigation indicates that this set is capable of finding good quality solutions to a variety of standard COPs efficiently. The transition operators are:

- *Move*: An element is moved from one list to the end of another list.
- *Swap*: The positions of two elements, from the same or different lists, are swapped.
- *Inversion*: The sequence between two elements on the same list is reversed.
- *Reposition*: The position of an element in a list is changed.
- *Add*: An element is added to a list.
- *Drop*: An element is removed from a list.
- *Change*: The value of a list element changes to another value.

Search may proceed either through feasible or infeasible space in order to locate solution states. For list-based problems, there are effectively two spaces that are bound by the list and problem constraints respectively. The transition operators can be applied so that the list constraints are satisfied and this is determined by the characteristics of the problem's list based model. There are a number of rules that govern the suitability of transition operators so that the problem's list constraints are not violated as shown in Table 1. Note, as in Section 2.1,  $y$  represents any list structure within list  $x$ .

### *Desired location of Table 1*

A transition is made in accordance with the rules of the particular meta-heuristic technique, subject to the list constraints being satisfied. For instance, if a value of a particular element appears once on list  $y$ , the constraint  $\text{min\_count}(y) = 1$  makes it infeasible to drop this element.

There are two ways of dealing with problem constraints in order to ensure that feasible solutions are obtained. These are *Feasibility Maintenance* and *Feasibility Restoration*. The first method allows only transitions that do not violate the problem constraints. For instance, in the GAP a job is only moved to a new sub-list (i.e. an agent) if that agent has the enough spare capacity to perform that job. Feasibility restoration on the other hand initially performs a transition without regard to the problem constraints and then employs a number of feasibility maintaining transitions in order to obtain problem feasibility once more. The advantage of the former technique is that it is computationally inexpensive to implement. However, feasibility restoration can traverse infeasible space and is required if the problem is tightly constrained and a straight feasibility maintaining transition is not possible. It has been determined that feasibility maintaining transitions are effective across the range of test problems and these will be used here.

## **3.1 Applying the Transition Operators**

Multiple transition operator neighbourhoods can be explored in the course of solving a particular problem. In many cases, not all possible operators (as per Table 1) are required, for instance one may be interested in the effect of performing only swaps for a TSP. Therefore the unwanted transition operators can be eliminated. The meta-heuristic technique is then performed using only those operators that have not been excluded, referred to as the *active operators*. At each iteration of the search, an operator is chosen from those in the active set. If all but one of the operators is eliminated, this operator is used at each iteration. In the case that there are two or more active operators, each operator is assigned a probability of being selected at each transition (totaling 1). A uniform random number is then generated to determine the operator. For example if the operator move is given the probabil-

Figure 1: A linked list representation of a solution to a GAP.

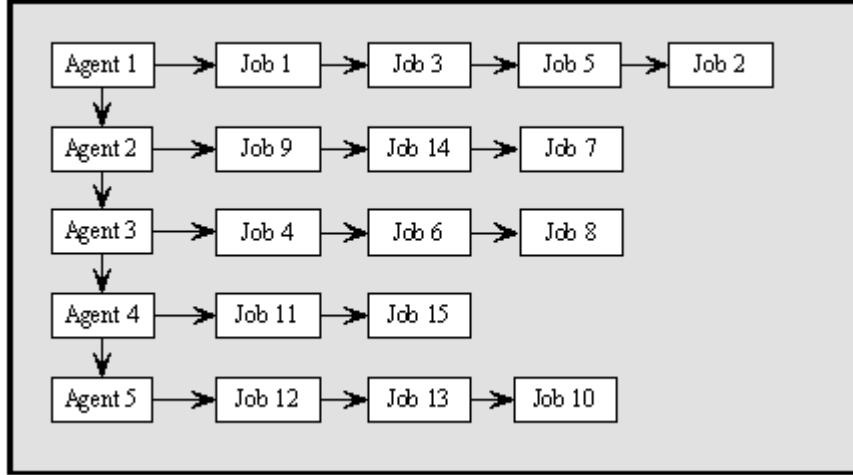


Table 1: Appropriate transition operators given a problem's list constraints.

| Count Constraints                              | List Size Constraints                                                             | Transition Operators                                                                                                                                |
|------------------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| $\min\_count(y) = 1$<br>& $\max\_count(y) = 1$ | $ y(i)  = N \quad \forall i \quad 1 \leq i \leq  y $ or<br>$ y  = 1$<br>$ y  > 1$ | swap, inversion, reposition<br>swap, inversion, reposition, move                                                                                    |
| $\max\_count(y) = 1$                           | $ y  = 1$<br>$ y  > 1$                                                            | swap, inversion, reposition, change,<br>add, drop<br>swap, inversion, reposition, change,<br>add, drop, move                                        |
| $\min\_count(y) = 1$ or<br>unspecified         | $ y  = 1$<br>$ y(i)  = N \quad \forall i \quad 1 \leq i \leq  y $<br>$ y  > 1$    | swap, inversion, reposition, change,<br>add, drop<br>swap, inversion, reposition, change<br>swap, inversion, reposition, change,<br>add, drop, move |

ity region  $[0,0.5]$  while swap is assigned  $(0.5,1]$  and 0.6 is the uniform random number, swap is chosen as the next transition operator.

### 3.2 Neighbourhood Size

Unlike SA, TS systematically examines the neighbourhood of the current solution to find a replacement solution. Many practical size problems have large neighbourhoods. Therefore the algorithm can become slow if the neighbourhood is large. In order that large problems can be solved in reasonable amounts of time, the system implements a simple candidate list strategy [24] based on the probabilistic selection of neighbours. For instance, if the probability is set to 1, then all neighbours are tested, however, if the probability is set to 0.5, then each neighbour has a 50% chance of being evaluated.

## 4 Implementation of the General Solver

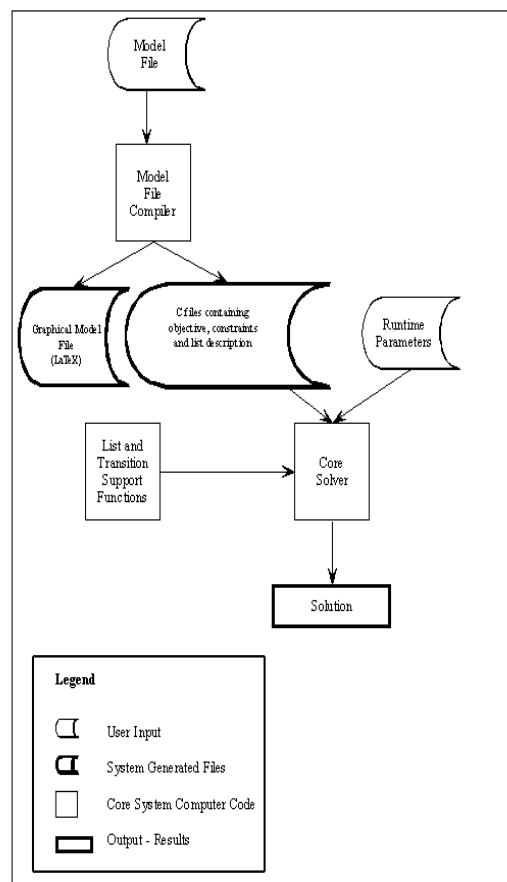
### 4.1 Solver Architecture

We have built a prototype system in order to test the approach described in the previous sections. Figure 2 shows the main components of the system, which include a compiler and a set of library files. A file containing an algebraic description of the problem model is read by a compiler and a set of code files in C is produced. In the process, the compiler also produces a mathematical list-based model in  $\text{\LaTeX} 2_{\epsilon}$  format using the algebraic list notation outlined in Section 2.1. These files contain the code that represents the objective function, list model, constraints and problem data. They are built using a standard C compiler and are linked with the solver library files. The latter contains all of the code necessary to implement the meta-heuristic algorithm which was chosen in the user supplied model. The output of the whole process is a purpose built executable program, which can then be run as a stand-alone code.

*Desired location of Figure 2*

The runtime system has two main tasks: to form an initial solution (see Section 4.3) to the problem as well as running the appropriate

Figure 2: Schematic of the prototype solver.



meta-heuristic search in order to produce a final solution to the problem being solved.

## 4.2 The Algebraic Modelling Language

The algebraic modelling language is based very loosely on the GAMS syntax [13], which is widely used and well understood. Whilst the syntax has been borrowed, semantic modifications have had to be made in order to account for the list modelling representation. Files can incorporate both data (for instance the distance matrix for the TSP) as well as objective and constraint functions. Alternatively, data and expressions can be contained in separate files.

An example of a file that describes a small GAP is presented in Figure 3. This file is divided into a number of distinct sections. The SCALAR section specifies problem constants, typically related to problem size. In this case, there are 5 agents (denoted M) and 15 jobs (denoted N). The SETS section allows range variables to be specified, i.e. J ranges in value from 1 to M (across the agents). The PARAMETER and TABLE sections contain the data for the problem instance. For the GAP, a cost matrix C, resource matrix, A, and an agent capacity vector B are required. The SOLUTION section is used to describe the list model and constraints. SIZE(X)=M denotes that there are M sublists while MIN\_COUNT(X)=1 and MAX\_COUNT(X)=1 indicate that each element can appear only once on the list. To specify that the value range is between 1 and N, the statement  $1 \leq X \leq N$  is used. Finally the EQUATIONS section allows the objective and constraints to be specified.

*Desired location of Figure 3*

Figure 4 shows the equivalent GAP formulation as a 0-1 ILP (the standard formulation of which is given in [11]). The main differences are that X is now a 0-1 matrix instead of a decision list and X is not used as index (as it is in the list modelling representation).

*Desired location of Figure 4*

Figure 3: Input file representing a GAP.

```

SCALAR
  N /15/
  M /5/;
SETS
  J /1-M/;
PARAMETER
  B(M) / 36 34 38 27 33 /;
TABLE
  C(M,N)
    / 17 21 22 18 24 15 20 18 19 18 16 22 24 24 16
      23 16 21 16 17 16 19 25 18 21 17 15 25 17 24
      16 20 16 25 24 16 17 19 19 18 20 16 17 21 24
      19 19 22 22 20 16 19 17 21 19 25 23 25 25 25
      18 19 15 15 21 25 16 16 23 15 22 17 19 22 24 /
  A(M,N)
    / 8 15 14 23 8 16 8 25 9 17 25 15 10 8 24
      15 7 23 22 11 11 12 10 17 16 7 16 10 18 22
      21 20 6 22 24 10 24 9 21 14 11 14 11 19 16
      20 11 8 14 9 5 6 19 19 7 6 6 13 9 18
      8 13 13 13 10 20 25 16 16 17 10 10 5 12 23 /;
SOLUTION
  SIZE(X)=M
  MIN COUNT(X)=1
  MAX COUNT(X)=1
  1<=X<=N;
EQUATIONS
  COST.. SUM(SUM(C(J,X(J,K)),K=1,SIZEOF(X(J))),J);
  CAPACITY(J): SUM(A(J,X(J,K)),K=1,SIZEOF(X(J)))<=B(J);
SOLVE

```

Figure 4: Input file representing a GAP (0-1 ILP notation).

```

SCALAR
  N /15/
  M /5/;
SETS
  J /1-M/;
  K /1-N/;
PARAMETER
  B(M) / 36 34 38 27 33 /;
TABLE
  C(M,N)
  / 17 21 22 18 24 15 20 18 19 18 16 22 24 24 16
    23 16 21 16 17 16 19 25 18 21 17 15 25 17 24
    16 20 16 25 24 16 17 19 19 18 20 16 17 21 24
    19 19 22 22 20 16 19 17 21 19 25 23 25 25 25
    18 19 15 15 21 25 16 16 23 15 22 17 19 22 24 /
  A(M,N)
  / 8 15 14 23 8 16 8 25 9 17 25 15 10 8 24
    15 7 23 22 11 11 12 10 17 16 7 16 10 18 22
    21 20 6 22 24 10 24 9 21 14 11 14 11 19 16
    20 11 8 14 9 5 6 19 19 7 6 6 13 9 18
    8 13 13 13 10 20 25 16 16 17 10 10 5 12 23 /;
VARIABLES
  X(J,K)
  TotalCost;
BINARY VARIABLE X;
EQUATIONS
  COST
  CAPACITY(J);
COST.. TotalCost =E= SUM(J,SUM(K,C(J,K)*X(J,K)))
CAPACITY(J).. SUM(K,A(J,K)*X(J,K)) =L= B(J);
MODEL GAP /ALL/;
SOLVE GAP USING MIP MINIMIZING TotalCost;

```

A full description of the new language is given in Appendix D of [46].

### 4.3 Generating an Initial Solution

The system described here uses a two-stage process in order to form initial feasible solutions. The first stage attempts to satisfy the list constraints while the second is concerned with obtaining the feasibility of the problem constraints. Once the two sets of feasibility have been established, the system invokes the meta-heuristic search algorithm.

The first stage of the algorithm consists of the following process. Initially, a set of feasible element values is produced, according to the *Count* and *Value Range* constraints. Following this, the element values are distributed throughout the list structure to ensure the length bounds on the sub-lists are valid. Accordingly, a solution that satisfies the list constraints is established. In many cases such as the TSP and QAP that have no problem constraints, the initial solution is feasible and the search process can begin immediately.

If problem constraints are present, the next stage is performed. As the constraints are arbitrary algebraic equations, it is difficult to devise a general algorithm that modifies the generated solution so that it is feasible. However, the meta-heuristic search engine can be applied to establish a feasible solution by allowing it to minimise the sum of the constraints' violation. When this sum is equal to 0, a feasible solution has been obtained. Constraint violation is calculated according to the relational operators that are present in the constraints. For instance, if the sign of a constraint is  $\leq$  and the left-hand side is larger than the right-hand side, the net difference is the amount of constraint violation. The constraint violations for the other signs  $\{<, >, \geq, =, \neq\}$  are calculated in a similar manner.

## 5 Methodology and Results

The system discussed in Section 4 has been tested on many standard problems from the literature. Because there are a large number of parameters that can be set, a full test of its performance is an impracticable task. Our main objective is to choose a suitable subset of problems, problem instances and parameters to determine the effectiveness of the system.

There are a number of ways that problems can be modeled using list notation and as such, we use list models that are best suited to iterative search. These are models that provide undulating cost landscapes rather than flatter surfaces. This is quite a common approach that has been used in many studies, including [6] [15] [32] [33]. For instance, there are a number of different formulations of the bin packing problem. The formulation we adopt minimises the excess weight for a given number of bins rather than minimising the actual number of bins. This is because the latter model has many equivalent packings (effectively flattening the search surface) that can lead to unproductive wandering through search space. In this paper, the graph and bin problems are modeled in this way. Therefore, in Tables 4 and 6 the optimal cost becomes 0 for a particular number of bins/colours (given in Table 2).

The computing platform used to perform the experiments is an IBM SP2 consisting of 22 RS6000 model 590 processors with a peak performance of 266 MFLOPs per node.

## 5.1 Problems Classes and Instances

Table 2 describes each of the COPs used to test the general system. The **Problem** heading gives the broad name of the problem while **Instance** is the name that is used to refer to a particular problem. The **Description** column provides a brief description of the size of the problem instance. The last two columns indicate the **Optimal Cost** (if proven optimality has been obtained in the literature) and the **Best-Known Cost** (otherwise).

*Desired location of Table 2*

## 5.2 Parameter Settings

The general system uses SA and TS search engines. Like other search engines, these require some parameter tuning. In this study, we attempt to do this in a systematic way. The performance of the system is tested by varying the following key groups of parameters:

Table 2: Problem classes and instances that are used in this study. Note: the multiple knapsack problem is a maximisation problem while the rest are minimisation problems.

| Problem                                                  | Problem Sources | Instance                                                                         | Description                                                                                                                                                     | Optimal Cost                                    | Best-Known Cost                       |
|----------------------------------------------------------|-----------------|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|---------------------------------------|
| Car Sequencing Problem (CSP)                             | [50]            | n20t1<br>n40t1<br>n60t1<br>n80t1<br>n80t2                                        | 20 cars, class 1<br>40 cars, class 1<br>60 cars, class 1<br>80 cars, class 1<br>80 cars, class 2                                                                |                                                 | 58<br>146<br>238<br>330<br>215        |
| Bin Packing Problem (BIN)                                | [9]             | bin1a1<br>bin1a2<br>bin2a1<br>bin2a2<br>bin3a1<br>bin3a2                         | 120 items<br>120 items<br>250 items<br>250 items<br>500 items<br>500 items                                                                                      | 48<br>49<br>99<br>100<br>198<br>201             |                                       |
| Graph Colouring Problems (GRAPH)                         | [33]            | C125.1<br>C125.5<br>C125.9<br>C250.1<br>C250.5<br>C250.9                         | 125 nodes, 736 edges<br>125 nodes, 3891 edges<br>125 nodes, 6961 edges<br>250 nodes, 3218 edges<br>250 nodes, 15668 edges<br>250 nodes, 27897 edges             |                                                 | 5<br>17<br>43<br>8<br>29<br>71        |
| Uncapacitated Single Allocation P Hub Problem(USApHMP)   | [21]            | ap20a2<br>ap20a3<br>ap25a2<br>ap25a3                                             | 20 nodes, 2 hubs<br>20 nodes, 3 hubs<br>25 nodes, 2 hubs<br>25 nodes, 3 hubs                                                                                    | 172816.7<br>151533.1<br>175542<br>155256.3      |                                       |
| Uncapacitated Multiple Allocation P Hub Problem(UMApHMP) | [21]            | ap10a2<br>ap10a3<br>ap10a4<br>ap10a5                                             | 20 nodes, 2 hubs<br>20 nodes, 3 hubs<br>20 nodes, 4 hubs<br>20 nodes, 5 hubs                                                                                    | 163603.94<br>131581.79<br>107354.73<br>86028.88 |                                       |
| Quadratic Assignment Problem(QAP)                        | [14]            | nug20<br>nug30<br>tho40<br>esc64a<br>sko72<br>will00                             | 20 facilities/locations<br>30 facilities/locations<br>40 facilities/locations<br>64 facilities/locations<br>72 facilities/locations<br>100 facilities/locations | 1285<br>3062                                    | 120258<br>58<br>33158<br>136522       |
| Travelling Salesman Problem(TSP)                         | [49]            | gr24<br>swiss42<br>brazil58<br>st70<br>kroA100<br>ch130                          | 24 cities<br>42 cities<br>58 cities<br>70 cities<br>100 cities<br>130 cities                                                                                    | 1272<br>1273<br>25395<br>675<br>21282<br>6110   |                                       |
| Generalised Assignment Problem(GAP)                      | [9]             | gapA5-100<br>gapA5-200<br>gapA10-100<br>gapA10-200<br>gapA20-100<br>gapA20-200   | 100 jobs, 5 agents<br>200 jobs, 5 agents<br>100 jobs, 10 agents<br>200 jobs, 10 agents<br>100 jobs, 20 agents<br>200 jobs, 20 agents                            | 1698<br>3235<br>1360<br>2623<br>1158<br>2339    |                                       |
| Graph Partitioning Problem(GPP)                          | [32]            | G124.04<br>G124.08<br>G124.16<br>G250.02<br>G250.04<br>G250.08                   | 124 nodes, 635 edges<br>124 nodes, 1240 edges<br>124 nodes, 2542 edges<br>250 nodes, 1224 edges<br>250 nodes, 2566 edges<br>250 nodes, 4842 edges               |                                                 | 63<br>178<br>449<br>114<br>357<br>828 |
| Multiple Knapsack Problem(MKP)                           | [9]             | weing1<br>pb6<br>sent01<br>sent02<br>weish25<br>weish29                          | 28 items, 2 knapsacks<br>40 items, 30 knapsacks<br>60 items, 30 knapsacks<br>60 items, 30 knapsacks<br>80 items, 5 knapsacks<br>90 items, 5 knapsacks           | 141278<br>776<br>7772<br>8722<br>9939<br>9410   |                                       |
| Maximum Clique Problem(MCP)                              | [8]             | johnson16-2-4<br>keller4<br>c-fat200-1<br>brock200_2<br>brock200_1<br>brock200_3 | 120 nodes, 5460 edges<br>171 nodes, 9435 edges<br>200 nodes, 1534 edges<br>200 nodes, 9876 edges<br>200 nodes, 14834 edges<br>200 nodes, 12048 edges            | 8<br>12                                         | 11<br>12<br>21<br>15                  |
| Time Tabling Problem (TTP)                               | [2]             | hdt4<br>hdt5<br>hdt6<br>hdt7<br>hdt8                                             | 120 tuples, 30 periods<br>150 tuples, 30 periods<br>180 tuples, 30 periods<br>210 tuples, 30 periods<br>240 tuples, 30 periods                                  | 0<br>0<br>0<br>0<br>0                           |                                       |

- The transition operators and the probability of applying these operators.
- The search engine specific parameters.

The parameters are set relatively coarsely. This is because the amount of computing time that would be involved in testing all combinations of parameter settings would be prohibitive. Therefore, the parameter settings outlined here are used to conduct tests that consist of runs involving small amounts of CPU time. Based on the outcome of these experiments, suitable parameter settings for different problem types can be derived.

The most time consuming set of experiments is the variation of the transition operator probabilities (See Section 3.1). Accordingly, a test set for each problem type has been devised and these are given in Table 3. For instance, the GAP has four settings in which the probabilities of the move and swap operators are  $\{\{1, 0\}, \{0.5, 0.5\}, \{0.2, 0.8\}, \{0.8, 0.2\}\}$  respectively. Each set of probabilities for a particular problem type is given a *transition set reference number* in order to uniquely identify the set in subsequent tables. Due to the large amount of computer time that could be required, a full exploration of transition probability parameter space (according to Table 3) will only occur for the SA runs. The best transition probability set obtained for a particular problem type with SA is used for the TS runs (this process is described in Section 5.4).

#### *Desired location of Table 3*

The following gives a brief description of the parameters required for the SA and TS engine.

**Simulated Annealing** The SA search engine implements Connolly's [19] [20] Q8-7 cooling schedule as it has been shown to be quite successful [6] [19] [20]. The cooling schedule is based on reheating the temperature a number of times throughout the search. Both the number of times that this reheating occurs as well as the interval between reheats can be altered. In this paper we use a reheating run length of  $\{1000, 5000, 10000\} \times (p_x - q_x)$ . The SA engine also has an

Table 3: Transition probability settings for each problem type.

| PROBLEM CLASS           | TRANSITION<br>SET REFERENCE<br>NUMBER |      |      |           |            |      |      |        |
|-------------------------|---------------------------------------|------|------|-----------|------------|------|------|--------|
|                         |                                       | Move | Swap | Inversion | Reposition | Add  | Drop | Change |
| CSP, GPP                | 1                                     | 1    |      |           |            |      |      |        |
| GAP, BIN,<br>GRAPH, TTP | 1                                     | 1    |      |           |            |      |      |        |
|                         | 2                                     | 0.5  | 0.5  |           |            |      |      |        |
|                         | 3                                     | 0.2  | 0.8  |           |            |      |      |        |
|                         | 4                                     | 0.8  | 0.2  |           |            |      |      |        |
| USApHMP                 | 1                                     | 1    |      |           |            |      |      |        |
|                         | 2                                     | 0.5  | 0.5  |           |            |      |      |        |
|                         | 3                                     | 0.2  | 0.8  |           |            |      |      |        |
|                         | 4                                     | 0.8  | 0.2  |           |            |      |      |        |
|                         | 5                                     | 0.4  | 0.4  | 0.2       |            |      |      |        |
| TSP, QAP                | 1                                     | 1    |      |           |            |      |      |        |
|                         | 2                                     |      |      |           |            |      |      |        |
|                         | 3                                     |      |      |           |            |      |      |        |
|                         | 4                                     | 0.33 |      | 0.33      | 0.33       |      |      |        |
|                         | 5                                     | 0.8  |      | 0.1       | 0.1        |      |      |        |
|                         | 6                                     | 0.1  |      | 0.8       | 0.1        |      |      |        |
|                         | 7                                     | 0.1  |      | 0.1       | 0.8        |      |      |        |
| MCP                     | 1                                     | 1    |      |           |            |      |      |        |
| MKP                     | 1                                     |      |      |           |            | 0.33 | 0.33 | 0.33   |
|                         | 2                                     |      |      |           |            | 0.5  | 0.5  | 0      |
|                         | 3                                     |      |      |           |            | 0.25 | 0.25 | 0.5    |
|                         | 4                                     |      |      |           |            | 0.1  | 0.1  | 0.8    |
|                         | 5                                     |      |      |           |            | 0.4  | 0.4  | 0.2    |
| UMApHMP                 | 1                                     | 0.2  | 0.2  |           |            | 0.2  | 0.2  | 0.2    |
|                         | 2                                     | 0.4  | 0.4  |           |            | 0.1  | 0.1  |        |
|                         | 3                                     | 0.8  |      |           |            | 0.1  | 0.1  |        |
|                         | 4                                     | 0.3  | 0.3  |           |            | 0.1  | 0.1  | 0.2    |
|                         | 5                                     |      |      |           |            | 0.33 | 0.33 | 0.33   |

option that allows it to perform as many reheats as possible in order to try to reach a given solution quality.

**Tabu Search** The TS engine allows the tabu list size to be set. This parameter governs the number of tabu search iterations that a particular transition stays tabu. In this paper, the length of the tabu list is set to an indicator of the problem size as this has been used with some success by [54] [55]. For instance: the tabu list size for gap1-1 (15 jobs) becomes 15 and hk48 (48 cities) is 48. The tabu list is implemented as a matrix as this approach has been adopted by a number of other researchers [42] [43] [54] and has been shown to be quite effective. The tabu list in our system records whether a particular element has been placed on a certain sub-list. In the case where position of an element on a sub-list is of importance (for instance the TSP and QAP), the tabu list records the element against the position number as well. We initially experimented with a tabu list based on hashing vectors as described in [59]. While these required only a small amount of memory, it was found that their use degraded the performance of the tabu search. The neighbourhood evaluation probabilities (see Section 3.2) are set at  $\{0.1, 0.5, 1.0\}$ .

### 5.3 Termination Conditions

Each run is given at most 1600 seconds of computing time (chosen due to the constraints of the computer’s job allocation system). If the optimal solution is known for a particular problem instance, the run is terminated once this has been reached.

### 5.4 Results

Due to the large amount of results that have been generated, only a summary of results is presented here. Extensive results can be found in [46].

Tables 4 and 6 display the best results obtained for every problem instance with the meta-heuristic engines. The best results correspond to the use of the best parameter settings. See Section 5.5 for a description of how the best parameter settings were found. The summary tables (4 and 6) are divided into two main sections, **Cost** and **Runtime**. For each section, the minimum (denoted **Min**), median

(denoted **Med**), maximum (denoted **Max**) and Inter Quartile Range (denoted **IQR**) are given (as each problem instance is run with 10 random seeds). Non-parametric descriptive statistics are used as the data are highly non-normally distributed. An  $\alpha = 0.05$  is used for all statistical analyses. The **Runtime** section records the amount of CPU time required to reach the best solution cost for that particular run.

*Desired location of Table 4*

In order to determine the most appropriate transition operator set for a particular problem type, the Kruskal-Wallis procedure is used. “Transition set reference number” was the independent (grouping) variable while objective cost was the dependent variable. For many of the problem types, a significant result was recorded which indicates that the choice of transition operator probabilities effects quality of solution. Table 5 shows the order of performance of the transition operator sets for each problem type<sup>1</sup>. For instance, in the case of BIN problems, transition set 2 (rank 1) was the best overall while set 1 (rank 4) was the worst. In the case of UMapHMP, sets 2,3,4 and 6 recorded equivalent objective costs and therefore “runtime” was used to distinguish between these sets.

The TS engine is run with the best transition operator set (rank order 1) for each problem type.

*Desired location of Table 5*

*Desired location of Table 6*

---

<sup>1</sup>Post-hoc analysis was used to determine whether there were any significant performance differences between the transition operator sets for each problem type. However, as it is difficult to construct overall ranks from this information, the Kruskal-Wallis median ranks were also taken into consideration.

Table 4: Summarised results of the SA engine.

| Problem | Instance      | Optimal/Best<br>Known Cost | Cost   |          |        |        | Runtime (seconds) |        |      |        |
|---------|---------------|----------------------------|--------|----------|--------|--------|-------------------|--------|------|--------|
|         |               |                            | Min    | Med      | Max    | IQR    | Min               | Med    | Max  | IQR    |
| CSP     | n20t1         | 58                         | 58     | 58       | 58     | 0      | 0                 | 7.5    | 27   | 6.25   |
|         | n40t1         | 146                        | 142    | 144      | 145    | 1      | 2                 | 450.5  | 1277 | 517    |
|         | n60t1         | 238                        | 228    | 231      | 232    | 1      | 1                 | 2      | 1232 | 702.25 |
|         | n80t1         | 330                        | 312    | 316      | 321    | 3.5    | 3                 | 6.5    | 447  | 4.75   |
| BIN     | bin1a1        | 0                          | 0      | 0        | 0      | 0      | 0                 | 1      | 2    | 0.5    |
|         | bin1a2        | 0                          | 0      | 0        | 0      | 0      | 0                 | 0.5    | 1    | 0.5    |
|         | bin2a1        | 0                          | 0      | 0        | 0      | 0      | 10                | 16     | 47   | 7      |
|         | bin2a2        | 0                          | 0      | 0        | 0      | 0      | 4                 | 7      | 9    | 2.5    |
|         | bin3a1        | 0                          | 0      | 0        | 0      | 0      | 62                | 170    | 413  | 85.5   |
|         | bin3a2        | 0                          | 0      | 0        | 0      | 0      | 173               | 241    | 370  | 104    |
| GRAPH   | C125.1        | 0                          | 0      | 2        | 2      | 0.75   | 14                | 100    | 409  | 176.25 |
|         | C125.5        | 0                          | 3      | 6        | 10     | 1.75   | 24                | 81     | 232  | 105    |
|         | C125.9        | 0                          | 1      | 2        | 3      | 0      | 46                | 179    | 253  | 26.5   |
|         | C250.1        | 0                          | 2      | 3.5      | 11     | 1.75   | 157               | 727    | 1430 | 597    |
|         | C250.5        | 0                          | 7      | 10       | 13     | 1      | 193               | 344.5  | 789  | 212.25 |
|         | C250.9        | 0                          | 7      | 10       | 13     | 1.5    | 472               | 728    | 883  | 255    |
| USApHMP | ap20a2        | 172817                     | 172817 | 172817   | 172817 | 0      | 36                | 52.5   | 95   | 26.25  |
|         | ap20a3        | 151533                     | 151533 | 151533   | 151533 | 0      | 39                | 61     | 121  | 31     |
|         | ap25a2        | 175542                     | 175542 | 175542   | 175850 | 308    | 383               | 649.5  | 1460 | 527.25 |
|         | ap25a3        | 155256                     | 155256 | 155611   | 159944 | 354.5  | 406               | 665    | 1228 | 345    |
| UMApHMP | ap10a2        | 163604                     | 163604 | 163604   | 163604 | 0      | 0                 | 0.5    | 1    | 0.25   |
|         | ap10a3        | 131582                     | 131582 | 131582   | 131582 | 0      | 0                 | 3      | 14   | 4      |
|         | ap10a4        | 107354                     | 107355 | 107355.5 | 107355 | 0      | 2                 | 14.5   | 57   | 21.25  |
|         | ap10a5        | 86029                      | 86029  | 86029    | 86029  | 0      | 0                 | 12.5   | 151  | 34     |
| QAP     | nug20         | 1285                       | 1285   | 1285     | 1285   | 0      | 4                 | 12     | 22   | 7      |
|         | nug30         | 3062                       | 3062   | 3062     | 3064   | 0      | 20                | 316.5  | 1527 | 866.5  |
|         | tho40         | 120258                     | 120535 | 121016.5 | 121903 | 552.25 | 81                | 465    | 1284 | 644.25 |
|         | esc64a        | 58                         | 58     | 58       | 58     | 0      | 0                 | 3      | 25   | 2.5    |
|         | sko72         | 33158                      | 33232  | 33307    | 33400  | 70.5   | 140               | 867    | 1517 | 1058   |
|         | wil100        | 136522                     | 136791 | 137048   | 137162 | 142.5  | 348               | 574    | 1150 | 243    |
| TSP     | gr24          | 1272                       | 1272   | 1272     | 1272   | 0      | 0                 | 4      | 8    | 1      |
|         | swiss42       | 1273                       | 1273   | 1273     | 1273   | 0      | 17                | 33.5   | 85   | 11     |
|         | brazil58      | 25395                      | 25395  | 25395    | 25395  | 0      | 373               | 1093   | 1473 | 643.25 |
|         | st70          | 675                        | 676    | 681.5    | 686    | 3.5    | 11                | 946    | 1122 | 179.25 |
|         | kroA100       | 21282                      | 6272   | 6395     | 6502   | 94.25  | 700               | 1217   | 1552 | 193    |
|         | ch130         | 6110                       | 21320  | 21600.5  | 22042  | 267    | 43                | 725.25 | 1292 | 1168   |
| GAP     | gapA5-100     | 1698                       | 1698   | 1700     | 1700   | 0.75   | 1                 | 2.5    | 236  | 2.75   |
|         | gapA5-200     | 3235                       | 3235   | 3235     | 3236   | 0.75   | 3                 | 11     | 21   | 7.25   |
|         | gapA10-100    | 1360                       | 1360   | 1361     | 1363   | 1      | 1                 | 2.5    | 5    | 2      |
|         | gapA10-200    | 2623                       | 2623   | 2626.5   | 2630   | 2      | 6                 | 15     | 46   | 18.5   |
|         | gapA20-100    | 1158                       | 1158   | 1158     | 1163   | 0      | 2                 | 4.5    | 8    | 2.25   |
|         | gapA20-200    | 2339                       | 2339   | 2339     | 2342   | 0      | 25                | 41.5   | 130  | 39.75  |
| GPP     | G124.04       | 63                         | 63     | 64       | 64     | 0.75   | 3                 | 231    | 340  | 60     |
|         | G124.08       | 178                        | 178    | 178      | 181    | 1.5    | 187               | 241    | 599  | 65.5   |
|         | G124.16       | 449                        | 449    | 449      | 449    | 0      | 166               | 187    | 356  | 151    |
|         | G250.02       | 114                        | 114    | 117      | 120    | 3.5    | 928               | 1041.5 | 1407 | 244    |
|         | G250.04       | 357                        | 358    | 360      | 364    | 3.75   | 217               | 1026.5 | 1238 | 170.75 |
|         | G250.08       | 828                        | 828    | 831      | 837    | 5.5    | 660               | 972    | 1514 | 446.25 |
| MKP     | weing1        | 141278                     | 121931 | 140778   | 141278 | 613.75 | 1                 | 1.5    | 4    | 0.5    |
|         | pb6           | 776                        | 650    | 776      | 776    | 14     | 0                 | 3.5    | 9    | 4      |
|         | sent01        | 7772                       | 7772   | 7772     | 7772   | 0      | 2                 | 63     | 445  | 28     |
|         | sent02        | 8722                       | 8711   | 8720.5   | 8722   | 2      | 24                | 143    | 1163 | 910.5  |
|         | weish25       | 9939                       | 9788   | 9923     | 9939   | 16     | 25                | 123    | 535  | 298    |
|         | weish29       | 9410                       | 9244   | 9304     | 9410   | 135.25 | 10                | 49.5   | 160  | 66     |
| MCP     | johnson16-2-4 | 0                          | 2      | 2        | 2      | 0      | 0                 | 0      | 0    | 0      |
|         | keller4       | 0                          | 0      | 0        | 0      | 0      | 6                 | 13.5   | 46   | 7.25   |
|         | c-fat200-1    | 0                          | 1      | 1        | 1      | 0      | 9                 | 17.5   | 40   | 22.25  |
|         | brock200_2    | 0                          | 3      | 3        | 3      | 0      | 3                 | 60.5   | 115  | 40     |
|         | brock200_1    | 0                          | 3      | 3        | 3      | 0      | 84                | 139.5  | 229  | 75.25  |
|         | brock200_3    | 0                          | 1      | 1        | 1      | 0      | 3                 | 60.5   | 77   | 5      |
| TTP     | hdt4          | 0                          | 0      | 0        | 0      | 0      | 4                 | 14.5   | 31   | 9      |
|         | hdt5          | 0                          | 0      | 0        | 3      | 0      | 16                | 35     | 93   | 25.5   |
|         | hdt6          | 0                          | 0      | 0        | 3      | 1.5    | 40                | 106    | 334  | 63.25  |
|         | hdt7          | 0                          | 0      | 0        | 4      | 2.75   | 72                | 263    | 534  | 120.25 |
|         | hdt8          | 0                          | 0      | 2        | 4      | 3.75   | 176               | 485    | 694  | 231.5  |

## 5.5 Performance Analysis

The results show that despite the relatively short amount of computational time available for these runs, optimal and best-known solutions are frequently achieved (74% for the SA engine and 58% for the TS engine). In some cases (notably the CSP) many of the best-known solution costs have been well exceeded by both the SA and TS search engines.

Despite good performance on a range of problems, the general solver is not quite as efficient on some problem types, in particular GRAPH and MCP. In the case of GRAPH, we chose the smallest number of colours for each instance attempted by Johnson et al. [33]. These colourings could only be obtained by Johnson et al. [33] after many hours of computer processing time (and not at all for C125.9 and C250.9). The results indicate that our system produces close near optimal colourings.

For our particular implementation of the meta-heuristic search algorithms, it appears that SA generally outperforms TS. However, this is hardly a fair comparison because of two reasons a) different implementations of the algorithms will produce different results b) TS was run using only one transition set per problem type while SA was tested under a range of conditions.

The same technique that has been applied in Table 5 has been used in Tables 7 and 8 to determine the most effective SA cooling length and TS probabilistic candidate list setting respectively. For instance, a cooling length of 10000 generally outperforms lengths of 1000 and 5000 for the CSP. In order to make an overall assessment, the average rank of the cooling length and neighbourhood probability setting is produced. From this it can be concluded that:

- *SA*: Longer cooling lengths typically improve solution quality.
- *TS*: Evaluating half of the neighbours at each iteration is usually more effective than evaluating all of the neighbours.

*Desired location of Table 7*

## 6 Conclusions

The work contained in this paper has shown that a general COP solver that is based on a linked list modelling representation is an efficient means of solving traditional COPs. This approach is novel as a problem can be specified in a textual format and solved directly using meta-heuristic search engines. This is in contrast to the traditional method of developing problem-customised optimisation programmes. Hence, the amount of time taken from initial problem specification to the final solution is dramatically reduced. As such, we are satisfied that the system is a good test-bench for model prototypers and those who wish to test various meta-heuristic implementations in a standard way.

The system shows good performance on a wide range of problems, frequently reaching the optimal and best-known solutions. Where this is not the case, generally solutions with a few percent deviation are produced. The performance is dependent on the chosen transition operators and their associated probabilities. To a lesser extent, the performance of this implementation is influenced by runtime parameters such as the TS probabilistic candidate list strategy and SA reheating length.

This research was undertaken as part of a larger project that aims to develop application specific computers for solving COPs [3] [5] [6]. The advantage of these systems is that a specific hardware design has real time advantages over conventional workstations. In addition, we have also developed a parallel implementation of the tabu search engine [47] as well as a feasibility restoration scheme [46].

We would like to test the system on an expanded problem set with a greater range of meta-heuristic engines and parameter sets. In particular, we would like to produce a GRASP engine. This can be achieved by modifying the initial solution procedure (described in Section 4.3) to act as the construction phase of the algorithm. Additionally, we wish to incorporate other heuristic engines that are not necessarily based on iterative search or the use of local transition operators, such as GAs and ANNs. The solver will also be enhanced so that it can accept problem formulations that require more than one level of sub-

list (at the present time, only one level of sub-list is supported). The other limitations of the system are outlined in [46].

Another interesting avenue of research is to experiment with different list transition operators. The list operators used in this paper represent only very simple transitions. There is no reason why two or more operators could not be combined in order to produce more complex compound operators.

## References

- [1] D. Abramson, "Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms," *Management Science*, vol. 37, pp. 98-113, 1991.
- [2] D. Abramson, and H. Dang, "School Timetables: A Case Study in Simulated Annealing," *Applied Simulated Annealing, Lecture Notes in Economics and Mathematics Systems*, Springer-Verlag, Ed. V. Vidal, Chapter 5, pp. 103 - 124, 1993.
- [3] D. Abramson, A. de Silva, M. Randall and A. Postula, "Special Purpose Computer Architectures for High Speed Optimisation," in *Proceedings of the Second Australasian Conference on Parallel and Real Time Systems Conference*, Perth, Australia, 1995, pp. 13 - 20.
- [4] D. Abramson, H. Dang and M. Krishnamoorthy, "A Comparison of Two Methods for Solving 0-1 Integer Programs Using a General Purpose Simulated Annealing," *Annals of Operations Research*, vol. 63, pp. 129-150, 1996.
- [5] D. Abramson, P. Logothetis, M. Randall and A. Postula, "Application Specific Computers for Combinatorial Optimisation," in *Proceedings of The Australian Computer Architecture Workshop*, Sydney, Australia, pp. 29-43, 1997.
- [6] D. Abramson and M. Randall "A Simulated Annealing Code for General Integer Linear Programs," *Annals of Operations Research*, vol. 86, pp. 3-24, 1999.
- [7] T. Bäck, *A User's Guide to GENESYs 1.0*, University of Dortmund, Department of Computer Science, 1992.

- [8] R. Battiti and M. Protasi, "Reactive Local Search for the Maximum Clique Problem," International Computer Science Institute, Technical Report TR-95-052, 1995.
- [9] J. Beasley, "OR-library: Distributing Test problems by Electronic Mail," Journal of the Operational Research Society, vol. 41, pp. 1069-1072, 1990.
- [10] J. Beasley, M. Krishnamoorthy, D. Abramson, Y. Sharaiha, "Scheduling Aircraft Landing - The Static Case," Submitted to Operations Research.
- [11] J. Beasley and P. Chu, "A Genetic Algorithm for the Multiconstraint Knapsack Problem," Computers and Operations Research, vol. 24, pp. 17-23, 1997.
- [12] J. Beasley and P. Chu, "Constraint Handling in Genetic Algorithms: the Set Partitioning Problem," Journal of Heuristics, vol. 4, pp. 323-357, 1998.
- [13] A. Brooke, D. Kendrick, A. Meeraus and R. Raman, GAMS Language Guide, GAMS Development Corporation, 1997.
- [14] R. Burkard, S. Karisch and F. Rendl, "QAPLIB - A Quadratic Assignment Problem Library," Journal of Global Optimization, vol. 10, pp. 391-403, 1997.
- [15] M. Chams, A. Hertz and D. de Werra, "Some Experiments with Simulated Annealing for Coloring Graphs," European Journal of Operational Research, vol. 32, pp. 260-266, 1987.
- [16] S. Chanas and P. Kobylanski, "A New Heuristic Algorithm Solving the Linear Ordering Problem," Institute of Industrial Engineering and Management, Technical Report PL-50-372, 1995.
- [17] J. Chandy and B. Prithviraj, "Parallel Simulated Annealing Strategies for VSLI Cell Placement," in Proceedings of the 9<sup>th</sup> International Conference on VSLI Design, Bangalore, India, 1996.
- [18] P. Chu and J. Beasley, "A Genetic Algorithm for the Generalised Assignment Problem," Computers and Operations Research, vol. 24, pp. 17-23, 1997.
- [19] D. Connolly, "An Improved Annealing for the QAP," European Journal of Operational Research, vol. 46, pp. 93-100, 1990.
- [20] D. Connolly, "General Purpose Simulated Annealing," Journal of the Operational Research Society, vol. 43, pp. 495-505, 1992.

- [21] A. Ernst and M. Krishnamoorthy, "A Novel Exact Solution Approach Based on Shortest Paths for p-hub Median Problems," submitted to ORSA Journal on Computing, 1997.
- [22] T. Feo and M. Resende, "Greedy Randomised Adaptive Search Procedures," *Journal of Global Optimization*, vol. 51, pp. 109-133, 1995.
- [23] P. Gilmore and R. Gomory, "A Linear Programming Approach to the Cutting Stock Problem," *Operations Research*, vol. 9, pp. 849-859, 1961.
- [24] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers: Boston MA, 1997.
- [25] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley: Reading, MA, 1989.
- [26] J. Grefenstette, "A User's Guide to GENISIS," Navy Center for Applied Research in Artificial Intelligence, Washington, D.C, 1987.
- [27] P. Hart, N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on SSC*, vol. 4, pp. 100-107, 1968.
- [28] J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Responsibilities," *Proceedings National Academy of Sciences*, vol. 79, pp. 2554-2558, 1982.
- [29] L. Ingber, "Simulated Annealing: Practice Versus Theory," *Computer Modelling*, vol. 18, pp. 29-57, 1993.
- [30] L. Ingber, "Adaptive Simulated Annealing (ASA): Lessons Learned," *Control and Cybernetics*, vol. 25, pp. 33-54, 1996.
- [31] J. Jaffar and M. Maher, "Constraint Logic Programming: A Survey," *Journal of Logic Programming*, vol. 19/20, pp. 503-585, 1996.
- [32] D. Johnson, C. Aragon, L. McGeogh and C. Scheveon, "Optimization by Simulated Annealing: An experimental Evaluation Pt I, Graph Partitioning," *Operations Research*, vol. 37, pp. 865-892, 1991.
- [33] D. Johnson, C. Aragon, L. McGeogh and C. Scheveon, "Optimization by Simulated Annealing: An experimental Evaluation Pt

II, Graph Colouring and Number Partitioning," *Operations Research*, vol. 39, pp. 378-406, 1991.

- [34] T. Kampke, "Simulated Annealing: Use of a New Tool in Bin Packing," *Annals of Operations Research*, vol. 16, pp. 327-332, 1988.
- [35] S. Kirkpatrick, D. Gelatt and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.
- [36] P. Kouvelis and W. Chiang, "A Simulated Annealing Procedure for Single Row Layout Problems in Flexible Manufacturing Systems," *International Journal of Production Research*, vol. 30, pp. 717-732, 1992.
- [37] J. Lauriere, "A Language and a Program for Stating and Solving Combinatorial Problems" *Artificial Intelligence*, vol. 10, pp. 29-127, 1978.
- [38] E. Lawler, J. Lenstra, A. Rinnoy Kan, and D. Shmoys, *The Traveling Salesman Problem: A guided Tour of Combinatorial Optimization*, Wiley: Chichester, 1985.
- [39] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, vol. 21, pp. 498-516, 1973.
- [40] J. Little and K. Darby-Dowman, "The Significance of Constraint Logic Programming to Operational Research," *Operational Research Tutorial Papers*, M. Lawrence and C. Wilsdon (eds.), Operational Research Society, Birmingham, 1995.
- [41] C. Nugent, T. Vollman, and J. Runl, "An Experimental Comparison of Techniques for the Assignment of Facilities to Locations," *Operations Research*, vol. 16, pp. 150-173, 1968.
- [42] I. Osman, "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem," *Annals of Operations Research*, vol. 41, pp. 421-451, 1993.
- [43] I. Osman, "Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches," *Operations Research Spektrum*, vol. 17, pp. 211-225, 1995.
- [44] I. Osman and J. Kelly, *Metaheuristics: Theory and Applications*, Kluwer Academic Publishers: Norwell MA, 1996.

- [45] C. Petersen, "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects," *Management Science*, vol. 13, pp. 736-750, 1967.
- [46] M. Randall, "A General Modelling Systems and Meta-heuristic based Solver for Combinatorial Optimisation Problems," PhD thesis, Faculty of Environmental Science, Griffith University, 1999. On line at <http://www.it.bond.edu.au/randall/general.pdf>
- [47] M. Randall and D. Abramson, "A General Parallel Tabu Search Algorithm for Combinatorial Optimisation Problems," School of Information Technology, Bond University, Technical Report TR99-03, Submitted to the *Journal of Heuristics*, 1999.
- [48] G. Reinelt, "The Linear Ordering Problem: Algorithms and Applications, Research and Exposition", *Research in Exposition and Mathematics*, vol. 8, Heldermann Verlag: Berlin, 1985.
- [49] G. Reinelt, "TSPLIB - a Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, pp. 376-384, 1991.
- [50] K. Smith, M. Palaniswami and M. Krishnamoorthy, "Traditional Heuristic Versus Hopfield Neural Network Approaches to a Car Sequencing Problem," *European Journal of Operational Research*, vol. 93, pp. 300-316, 1996.
- [51] S. Sofianopoulou, "The Process Allocation Problem: a Survey of the Application of Graph-Theoretic and Integer Programming Approaches," *Journal of the Operational Research Society*, vol. 43, pp. 407-413, 1992.
- [52] R. Sasic and J. Gu, "Fast Search Algorithms for the N-Queens Problem," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, pp. 1572-1576, 1991.
- [53] H. Taha, *Operations Research: An introduction*, Macmillan Publishing Company: New York, 5<sup>th</sup> Edition, 1992.
- [54] E. Taillard, "Robust Taboo Search for the Quadratic Assignment Problem," *Parallel Computing*, vol. 17, pp. 443-455, 1991.
- [55] E. Taillard, P. Badeau, M. Gendreau, F. Guertin and J. Potvin, "A Tabu Search for the Vehicle Routing Problem with Soft Time Windows," *Transportation Science*, vol. 31, pp. 170-186, 1997.

- [56] D. Tank and J. Hopfield, “‘Neural’ Computation of Decisions in Optimization Problems,” *Biological Cybernetics*, vol. 52, pp. 141-152, 1985.
- [57] P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, The MIT Press: Cambridge, MA, 1989.
- [58] L. van Laarhoven, and E. Aarts, *Simulated Annealing: Theory and Applications*, D Reidel Publishing Company: Dordrecht, 1987.
- [59] D. Woodruff and E. Zemel, “Hashing Vectors for Tabu Search,” *Annals of Operations Research*, vol. 41, pp. 123-137, 1993.

Table 5: Rank order of the transition sets for each problem type.

| Problem | Transition Set Reference Number |   |   |   |   |   |   | Significance<br>(P) |
|---------|---------------------------------|---|---|---|---|---|---|---------------------|
|         | 1                               | 2 | 3 | 4 | 5 | 6 | 7 |                     |
| CSP     | 1                               |   |   |   |   |   |   | -                   |
| BIN     | 4                               | 1 | 3 | 2 |   |   |   | .000                |
| GRAPH   | 2                               | 4 | 1 | 3 |   |   |   | .321                |
| USApHMP | 1                               | 5 | 2 | 4 | 3 |   |   | .000                |
| UMApHMP | 4                               | 3 | 2 | 1 | 5 |   |   | .000 (cost)         |
|         |                                 |   |   |   |   |   |   | .000 (runtime)      |
| QAP     | 1                               | 7 | 6 | 3 | 2 | 4 | 5 | .000                |
| TSP     | 7                               | 1 | 6 | 3 | 4 | 2 | 5 | .000                |
| GAP     | 4                               | 1 | 2 | 3 |   |   |   | .000                |
| GPP     | 1                               |   |   |   |   |   |   | -                   |
| MKP     | 3                               | 5 | 2 | 1 | 4 |   |   | .000                |
| MCP     | 1                               |   |   |   |   |   |   | -                   |
| TTP     | 4                               | 1 | 3 | 2 |   |   |   | .000                |

Table 6: Summarised results of the TS engine.

| Problem | Instance      | Optimal/ Best<br>Known cost | Cost   |          |        |         | Runtime (seconds) |        |      |         |
|---------|---------------|-----------------------------|--------|----------|--------|---------|-------------------|--------|------|---------|
|         |               |                             | Min    | Med      | Max    | IQR     | Min               | Med    | Max  | IQR     |
| CSP     | n20t1         | 58                          | 58     | 58       | 58     | 0       | 0                 | 1      | 6    | 1.25    |
|         | n40t1         | 146                         | 142    | 142      | 143    | 1       | 7                 | 679.5  | 1202 | 788.75  |
|         | n60t1         | 238                         | 227    | 228      | 229    | 0.75    | 71                | 680.5  | 1597 | 1050.25 |
|         | n80t1         | 330                         | 311    | 313      | 314    | 1       | 425               | 1174   | 1566 | 477     |
| BIN     | bin1a1        | 0                           | 0      | 0        | 0      | 0       | 2                 | 4.5    | 10   | 2       |
|         | bin1a2        | 0                           | 0      | 0        | 0      | 0       | 2                 | 3      | 6    | 2       |
|         | bin2a1        | 0                           | 0      | 0        | 0      | 0       | 68                | 135    | 381  | 63      |
|         | bin2a2        | 0                           | 0      | 0        | 0      | 0       | 33                | 40     | 55   | 9       |
|         | bin3a1        | 0                           | 0      | 4.5      | 14     | 2.75    | 883               | 1300.5 | 1477 | 202.5   |
|         | bin3a2        | 0                           | 4      | 6        | 9      | 3.75    | 914               | 1256.5 | 1552 | 270     |
| GRAPH   | C125.1        | 0                           | 2      | 5        | 6      | 1       | 73                | 928    | 1553 | 867.25  |
|         | C125.5        | 0                           | 4      | 6.5      | 9      | 1.75    | 252               | 1015   | 1557 | 690.25  |
|         | C125.9        | 0                           | 2      | 3.5      | 5      | 1.75    | 585               | 845.5  | 1283 | 459.25  |
|         | C250.1        | 0                           | 16     | 22       | 24     | 3.5     | 570               | 1186   | 1530 | 494     |
|         | C250.5        | 0                           | 14     | 24.5     | 28     | 5.5     | 284               | 1167   | 1564 | 317     |
|         | C250.9        | 0                           | 17     | 22       | 24     | 3.5     | 740               | 1307   | 1543 | 344.5   |
| USApHMP | ap20a2        | 172817                      | 172817 | 172817   | 172817 | 0       | 3                 | 17.5   | 83   | 22.5    |
|         | ap20a3        | 151533                      | 151533 | 151533   | 151533 | 0       | 15                | 31.5   | 191  | 75.25   |
|         | ap25a2        | 175542                      | 175542 | 175542   | 175850 | 147     | 100               | 756.5  | 1185 | 660.25  |
|         | ap25a3        | 155256                      | 155256 | 155256.5 | 155611 | 0       | 60                | 673    | 1486 | 572.75  |
| UMApHMP | ap10a2        | 163604                      | 163604 | 163604   | 163604 | 0       | 55                | 257    | 437  | 181     |
|         | ap10a3        | 131582                      | 131582 | 132596   | 133898 | 1681.75 | 537               | 947.5  | 1506 | 479     |
|         | ap10a4        | 107354                      | 107355 | 108307.5 | 110982 | 1096    | 61                | 549    | 1562 | 681.5   |
|         | ap10a5        | 86029                       | 86795  | 88182.5  | 89118  | 1480.5  | 27                | 446.5  | 1271 | 551     |
| QAP     | nug20         | 1285                        | 1285   | 1285     | 1285   | 0       | 1                 | 70     | 1373 | 229.25  |
|         | nug30         | 3062                        | 3062   | 3066.5   | 3078   | 6       | 248               | 950    | 1294 | 469     |
|         | tho40         | 120258                      | 120439 | 120630   | 120852 | 263.25  | 108               | 1061.5 | 1597 | 678     |
|         | esc64a        | 58                          | 58     | 58       | 58     | 0       | 4                 | 23.5   | 539  | 38      |
|         | sko72         | 33158                       | 33197  | 33348.5  | 33686  | 245.5   | 677               | 1430.5 | 1591 | 654.25  |
|         | wil100        | 136522                      | 137080 | 137396   | 137527 | 166.75  | 700               | 1495   | 1591 | 183.25  |
| TSP     | gr24          | 1272                        | 1272   | 1272     | 1272   | 0       | 0                 | 1.5    | 12   | 2.5     |
|         | swiss42       | 1273                        | 1273   | 1273     | 1273   | 0       | 7                 | 33.5   | 106  | 35.25   |
|         | brazil58      | 25395                       | 25395  | 25395    | 25395  | 0       | 41                | 524    | 1073 | 378.5   |
|         | st70          | 675                         | 675    | 681.5    | 690    | 6.5     | 321               | 983.5  | 1530 | 723     |
|         | kroA100       | 21282                       | 21484  | 21946.5  | 27941  | 842.75  | 660               | 1015   | 1491 | 351.5   |
|         | ch130         | 6110                        | 6287   | 6519     | 6814   | 79.75   | 886               | 1143.5 | 1343 | 311     |
| GAP     | gapA5-100     | 1698                        | 1698   | 1698     | 1699   | 0       | 23                | 170    | 937  | 346.5   |
|         | gapA5-200     | 3235                        | 3235   | 3235     | 3237   | 0.75    | 399               | 629    | 1011 | 424     |
|         | gapA10-100    | 1360                        | 1360   | 1361     | 1361   | 1       | 43                | 281    | 1306 | 208     |
|         | gapA10-200    | 2623                        | 2624   | 2625     | 2629   | 1.75    | 458               | 893    | 1423 | 296     |
|         | gapA20-100    | 1158                        | 1158   | 1159     | 1160   | 1       | 46                | 325    | 1166 | 309.5   |
|         | gapA20-200    | 2339                        | 2342   | 2343     | 2346   | 1.75    | 503               | 839.5  | 1437 | 297.75  |
| GPP     | G124.04       | 63                          | 63     | 64       | 64     | 0       | 27                | 229.5  | 1531 | 396.75  |
|         | G124.08       | 178                         | 178    | 178      | 179    | 0       | 180               | 874    | 1308 | 831.25  |
|         | G124.16       | 449                         | 449    | 449      | 457    | 6       | 104               | 288.5  | 533  | 192.25  |
|         | G250.02       | 114                         | 117    | 125      | 130    | 2.5     | 613               | 1467   | 1549 | 743.5   |
|         | G250.04       | 357                         | 365    | 376      | 388    | 8       | 648               | 1366.5 | 1598 | 422     |
|         | G250.08       | 828                         | 834    | 847      | 868    | 16.25   | 1316              | 1513   | 1599 | 121     |
| MKP     | weing1        | 141278                      | 141278 | 141278   | 141278 | 0       | 9                 | 39     | 83   | 54.25   |
|         | pb6           | 776                         | 776    | 776      | 776    | 0       | 1                 | 25.5   | 108  | 43.5    |
|         | sent01        | 7772                        | 7739   | 7772     | 7772   | 27.5    | 216               | 525.5  | 1441 | 642     |
|         | sent02        | 8722                        | 8587   | 8721     | 8722   | 0       | 315               | 882.5  | 1398 | 511.25  |
|         | weish25       | 9939                        | 9761   | 9828.5   | 9857   | 56.5    | 121               | 961.5  | 1473 | 678     |
|         | weish29       | 9410                        | 9053   | 9154.5   | 9345   | 150.75  | 164               | 904    | 1194 | 618.75  |
| MCP     | johnson16-2-4 | 0                           | 2      | 2        | 2      | 0       | 0                 | 0      | 0    | 0       |
|         | keller4       | 0                           | 0      | 0        | 0      | 0       | 0                 | 6      | 38   | 18.75   |
|         | c-fat200-1    | 0                           | 1      | 1        | 1      | 0       | 0                 | 5.5    | 55   | 21.5    |
|         | brock200_2    | 0                           | 3      | 3        | 3      | 0       | 2                 | 14     | 218  | 47      |
|         | brock200_1    | 0                           | 3      | 3        | 3      | 0       | 5                 | 370    | 1289 | 560     |
|         | brock200_3    | 0                           | 3      | 3        | 4      | 1       | 2                 | 58     | 891  | 124.25  |
| TTP     | hdt4          | 0                           | 0      | 0        | 2      | 0       | 120               | 557.5  | 1568 | 687.75  |
|         | hdt5          | 0                           | 2      | 2        | 4      | 0.75    | 136               | 407    | 1598 | 771     |
|         | hdt6          | 0                           | 2      | 6        | 9      | 2       | 774               | 1231   | 1554 | 391.5   |
|         | hdt7          | 0                           | 4      | 11       | 16     | 5       | 679               | 1351.5 | 1567 | 270.75  |
|         | hdt8          | 0                           | 13     | 16.5     | 22     | 4.75    | 701               | 1368.5 | 1553 | 328.25  |

Table 7: Rank order of the different SA cooling lengths for each problem type.

| Problem      | SA Cooling Length |      |       | Significance<br>(P) |
|--------------|-------------------|------|-------|---------------------|
|              | 1000              | 5000 | 10000 |                     |
| CSP          | 3                 | 2    | 1     | .020                |
| BIN          | 3                 | 2    | 1     | .000                |
| GRAPH        | 3                 | 2    | 1     | .000                |
| USApHMP      | 3                 | 1    | 2     | .000                |
| UMApHMP      | 3                 | 2    | 1     | .007                |
| QAP          | 2                 | 1    | 3     | .620                |
| TSP          | 3                 | 2    | 1     | .006                |
| GAP          | 3                 | 2    | 1     | .080                |
| GPP          | 2                 | 1    | 3     | .000                |
| MKP          | 3                 | 2    | 1     | .000                |
| MCP          | 3                 | 2    | 1     | .150                |
| TTP          | 3                 | 2    | 1     | .000                |
| Average Rank | 2.83              | 1.75 | 1.42  |                     |

Table 8: Rank order of the different TS neighbourhood probabilities for each problem type.

| Problem      | Neighbour Probability |     |      | Significance<br>(P) |
|--------------|-----------------------|-----|------|---------------------|
|              | 0.1                   | 0.5 | 1    |                     |
| CSP          | 2                     | 1   | 3    | .014                |
| BIN          | 1                     | 2   | 3    | .002                |
| GRAPH        | 2                     | 1   | 3    | .049                |
| USApHMP      | 3                     | 1   | 2    | .000                |
| UMApHMP      | 1                     | 2   | 3    | .002                |
| QAP          | 1                     | 2   | 3    | .000                |
| TSP          | 3                     | 1   | 2    | .000                |
| GAP          | 3                     | 2   | 1    | .000                |
| GPP          | 1                     | 2   | 3    | .000                |
| MKP          | 2                     | 1   | 3    | .000                |
| MCP          | 1                     | 2   | 3    | .000                |
| TTP          | 3                     | 1   | 2    | .025                |
| Average Rank | 1.92                  | 1.5 | 2.58 |                     |